VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití smartphonu pro bezdrátové ovládání měřícího robotu. Measuring Robot Control by Smartphone

2015 Lukáš Dědek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Dědek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití smartphonu pro bezdrátové ovládání měřícího robotu.
Measuring Robot Control by Smartphone

Jazyk vypracování: čeština

Zásady pro vypracování:

Navhněte aplikaci pro mobilní telefon s OS Android, pro dálkové ovládání robotického podvozku přes WiFi síť s využitím soketového rozhraní. Pro ovládání směru a rychlosti budou využity polohové senzory integrované v MT.

1. Seznamte s fungováním a postupem měření polohových čidel používaných v mobilních telefonech s OS Android.
2. Seznamte se s programovým rozhraním OS Android pro přístup k senzorům, vytvořte testovací aplikaci.
3. Navrhněte komunikační rozhraní se zpětnou vazbou pro přenos dat z MT do řídícího počítače robotického podvozku a zpět.
4. Realizujte programové vybavení pro OS Android i pro Linux řídícího počítače. Na MT zobrazujte stav robotu.
5. Proveďte testování odezvy systému, jeho spolehlivost a stabilitu.
6. Shrňte získané zkušenosti a navrhněte možnosti dalšího rozvoje.

Seznam doporučené odborné literatury:

[1] Android SDK: http://developer.android.com/sdk/
[2] Linux Začínáme programovat, Richard Stones, Neil Metthew, COMPUTER PRESS, ISBN 80–7226–307–2

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

„Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal."

V Ostrave 5.5.2015 .....

.....

Lukáš Dědek

Poďakovanie

Rád by som sa poďakoval vedúcemu diplomovej práce Ing. Petrovi Olivkovi, Ph.D.,
za konzultácie a odbornú pomoc pri vytvorení tejto práce.

Abstrakt

Obsahom tejto diplomovej práce je popis vývoja aplikácie na riadenie robotického podvozka. Na užívateľské rozhranie je využitý mobilný telefón s operačným systémom Android a pri riadení využíva polohové snímače. Riadi sa upravený robotický vysávač Roomba od spoločnosti iRobot. Na sprostredkovanie riadenia sa využíva jednodoskový počítač Raspberry Pi komunikujúci pomocou Wi-Fi s mobilným telefónom a sériovou linkou s robotickým podvozkom.

Kľúčové slová

Android, Raspberry Pi, iRobot Roomba, socket, senzory, RS-232

Abstract

Contents of this thesis is to describe the development of applications for managing robotic chassis. On the user interface is used a cell phone with the Android operating system and uses the management position sensors. This cell phone manages the customized robot Roomba from iRobot company. To convey management is used single board computer Raspberry Pi communicating via Wi-Fi to a mobile phone and a serial line with robotic chassis.

Keywords

Android, Raspberry Pi, iRobot Roomba, socket, senzors, RS-232

Zoznam použitých skratiek

API	Application Programing Interface
DHCP	Dynamic Host Configuration Protocol
FET	Field Effect Transistor
GPS	Global Positioning System
MENS	Micro-Electro-Mechanical Systems
MT	Mobilný telefón
Os	Operačný systém
RWM	Read-Write-Memory
RPR	Robotický podvozok Roomba
TCP/IP	Transmission Control Protocol/Internet Protocol

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Obsah

1 Úvod

V ostatných rokoch dochádza k rozmachu vo vývoji a v komerčnom využití robotických zariadení na trhu, ako sú automatizované robotické vysávače, kosačky, umývacie zariadenia a lietajúce drony. Tento rozvoj a dostupnosť zariadení ponúka možnosť využiť ich aj na iné účely, aké boli zamýšľané pri ich vývoji. Jedným z hlavných dôvodov používania týchto zariadení je cena, ktorá je oproti vývoju nových zariadení oveľa nižšia.

Témou diplomovej práce je spracovanie softvéru, ktorý by zabezpečil riadenie robotického podvozka s využitím mobilného telefónu a jeho pohybových senzorov. Na účely diplomovej práce bolo určené zariadenie pochádzajúce od spoločnosti iRobot. Táto firma, jedna z prvých, ktorá začala predávať robotické vysávače, dnes predáva vysávače s najlepšími parametrami a recenziami. Ako operačný systém je zvolený Android, pretože má v aktuálnom čase najväčšie pokrytie na trhu a programátorom poskytuje operačný systém s otvoreným kódom.

V tejto diplomovej práci sú postupne opísané základné vlastnosti senzorov a ich spôsob používania v mobilnej aplikácii, ako aj potrebné hardvérové zariadenia na vytvorenie bezdrôtovej komunikácie medzi mobilným telefónom a robotickým vysávačom Roomba od spoločnosti iRobot. Práca obsahuje kompletné softvérové riešenie riadenia, postup jeho vývoja, testovanie odozvy a spoľahlivosti.

2 Polohové senzory v mobilných telefónoch

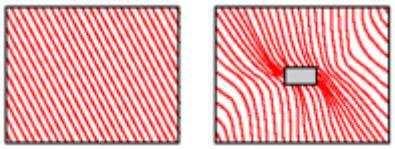
Moderné mobilné telefóny využívajú na zistenie polohy zariadenia veľké množstvo snímačov, ktoré poskytujú informácie o jeho polohe a okolí. Ide o tieto typy snímačov:

- **akcelerometer** – meria, s akým zrýchlením sa zariadenie pohybuje v troch osiach,
- **barometer** – meria hodnotu atmosférického tlaku a pomáha tiež zrýchliť výpočet nadmorskej výšky,
- **GPS** – určuje zemepisnú polohu na základe údajov získaných z družíc,
- **gyroskopický senzor** – udáva rýchlosť otáčania zariadenia v troch osiach,
- **magnetický senzor** – využíva sa na určenie magnetického severu,
- **proximity senzor** – zisťuje výskyt určitého predmetu pred senzorom, najčastejšie pomocou infračerveného žiarenia.

Pri využití mobilného telefónu na riadenie meracieho robota je vhodné použiť tri druhy snímačov. V nasledujúcich podkapitolách sú opísané základné princípy fungovania týchto snímačov a ich možné využitie.

2.1 Magnetický senzor

Základnou časťou magnetických senzorov je magnetorezistívny senzor. Jeho základ tvorí pliešok vyrobený z feromagnetického materiálu. Zložený je zo zmesi niklu a železa. Tento pliešok je zmagnetizovaný a v prípade nulového pôsobenia externých magnetických polí má pri danom prúde určitý odpor R. Pri pôsobení externých magnetických síl dochádza k zmene odporu na pliešku. Pri konštantnom prúde stačí merať rozdiely napätia a vyhodnotiť ich.
Nie je možné určiť orientáciu pôsobenia magnetického poľa. Preto nemožno nerozoznať sever od juhu. Táto nepresnosť sa rieši tzv. mostovým zapojením, pri ktorom sú využité dva mosty uložené kolmo na seba.



Obrázok 1: Pôsobenie zemského magnetického poľa na senzor [11]

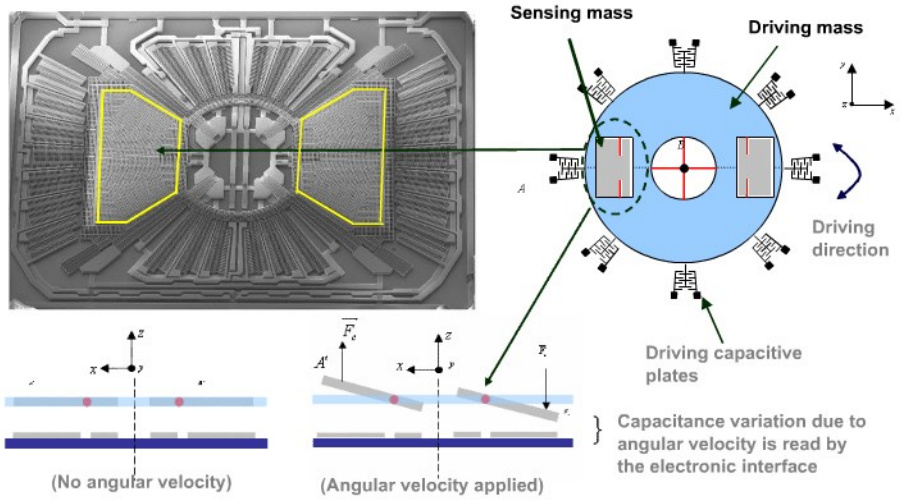
V mobilných telefónoch sa používajú maximálne tri mostíky uložené v troch osiach. Doplnené sú o polohový snímač, ktorý určuje sklon telefónu k rovine zemského povrchu. Tento snímač ako jediný senzor poskytuje informácie o magnetickom severe, ak sa zariadenie nepohybuje. Možno ho využiť aj na meranie intenzity magnetického poľa. [11]

2.2 Akcelerometer

Je jedným zo základných pohybových senzorov v mobilných telefónoch. Meria pri pohybe telefónu vibrácie alebo zrýchlenie. Akcelerometre v mobilných telefónoch fungujú na princípe piezoelektrického javu, čo je schopnosť kryštálu generovať elektrické napätie pri jeho deformovaní. Toto elektrické napätie je pritom dobre merateľné, pretože hmota kryštálu zostáva nezmenená a je vytvorený elektrický náboj úmerný zrýchleniu a sile, ktorá ho vygenerovala.
V mobilných telefónoch sa používa akcelerometer zabudovaný do obvodu telefónu. Jeho elektrický náboj je cez FET tranzistor prevedený na napäťový výstup s nízkou impedanciou. Impedancia je dobre merateľná bežnými meracími prístrojmi. [11]

2.3 Gyroskopický senzor

Patrí do skupiny MENS senzorov. Znamená to, že ide o elektromechanickú súčiastku. Konkrétne hovoríme o miniatúrne pohyblivých krídelkách nachádzajúcich sa v čipe, ktoré sa pri pohybe nakláňajú. Tým sa mení vzdialenosť od podkladových plôch, čo spôsobuje zmenu kapacity, ktorá sa následne meria. V mobilných zariadeniach sa používajú gyroskopy merajúce v troch osiach a ich čipy obsahujú tri krídelká. Gyroskop meria uhlovú rýchlosť na rozdiel od akcelerometra, ktorý meria zrýchlenie. Preto je výhodné použiť dáta z obidvoch senzorov na zvýšenie presnosti merania. [12]



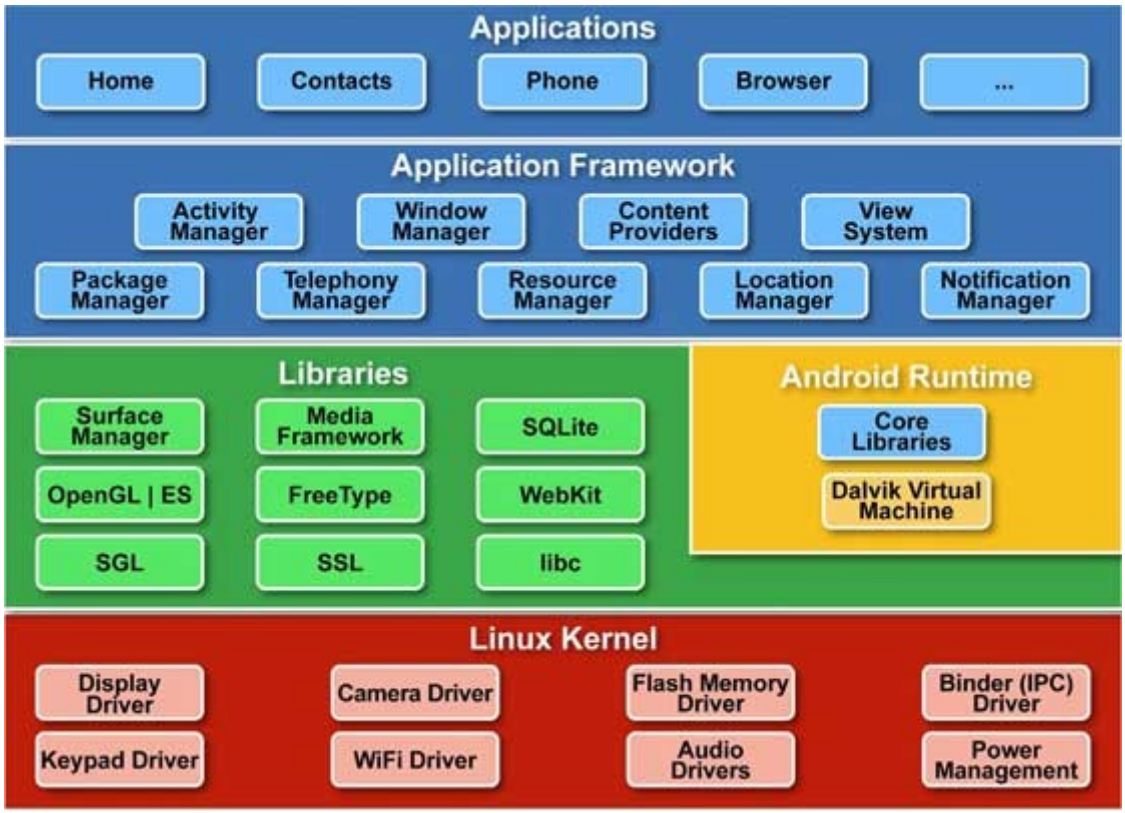
Obrázok 2: Princíp fungovania gyroskopu [12]

3 Použitie senzorov v operačnom systéme Android

Android je platforma s otvoreným zdrojovým kódom pre mobilné zariadenia a tablety, ktorej základ tvorí pozmenené Linuxové jadro. Systém vyvíjala od roku 2003 firma Android Inc. Potom ju roku 2005 kúpil Google a v septembri 2008 predstavil prvú komerčnú verziu operačného systému pod označením Android 1.0 (API level 1). V tejto verzii bolo možné využívať akcelerometer a magnetický senzor. Roku 2010 pribudol aj gyroskopický senzor vo verzii 2.3.

3.1 Architektúra osí Android

Architektúra operačného systému Android sa skladá z piatich vrstiev. Každá z týchto vrstiev vykonáva určité operácie a používajú sa viac-menej samostatne. Vrstvy nie sú od seba oddelené, a preto programátor môže využiť spoluprácu jednotlivých častí.



Obrázok 3: Architektúra operačného systému Android [6]

3.1.1 Linux Kernel

Jadro predstavuje najdôležitejšiu časť systému, pretože zaisťuje komunikáciu medzi hardvérovou a softvérovou časťou. Jeho hlavnú časť tvoria ovládače zabezpečujúce komunikáciu s jednotlivými časťami hardvéru a riadia procesy, ktoré sú uvedené v červenej časti obrázka. Ako už bolo napísané, Android nemá vyvinuté vlastné jadro, ale používa jadro operačného systému Linux, a to vo verzii 2.6.

3.1.2 Libraries

Knižnice zabezpečujúce základné funkcie systému sú napísané v programovacom jazyku C a C++.

- Surface manager – zobrazenie aplikácií a ich vrstiev.
- Open GL – práca s 3D grafikou.
- SGL – práca s 2D grafikou.
- Media Framework – práca s mediálnymi súbormi.
- FreeType – vykresľovanie písma.
- SSL – šifrovanie dát komunikácie.
- SQLite – práca s dátami.
- Webkit – vykreslenie webových stránok.
- Libc – základné knižnice jazyku C.

3.1.3 Android Runtime

V tejto vrstve sa nachádza Dalvik Virtual Machine, čo je virtuálny stroj, ktorý sa stará o prevod Java kódu; v tomto kóde sú prevedené aplikácie do natívneho kódu. Dalvik Virtual Machine nahrádza Java Virtual Machine z licenčných dôvodov a používa sa do verzie 4.3, kde je nahradený Dalvik Turbo. Ten je významne rýchlejší a úspornejší, pričom si zachováva spätnú kompatibilitu. V tejto vrstve sa tiež nachádzajú knižnice pre jazyk Java.

3.1.4 Application Framework

V tejto vrstve sa nachádzajú knižnice pre jazyk Java a poskytujú programátorovi využívať funkcie systému. Knižnice tvoria systémové API. Vďaka politike otvoreného zdrojového kódu má programátor prístup k rovnakým knižniciam ako operačný systém. To prispieva k tvorbe kvalitných aplikácií.

3.1.5 Aplications

V tejto vrstve sú aplikácie napísané v jazyku Java a slúžia pre používateľa. Patria sem:

- základné aplikácie na telefonovanie a písanie sms,
- klávesnica,
- aplikácie stiahnuté z Google Play,
- aplikácie tretích strán.

3.2 Vývoj aplikácií

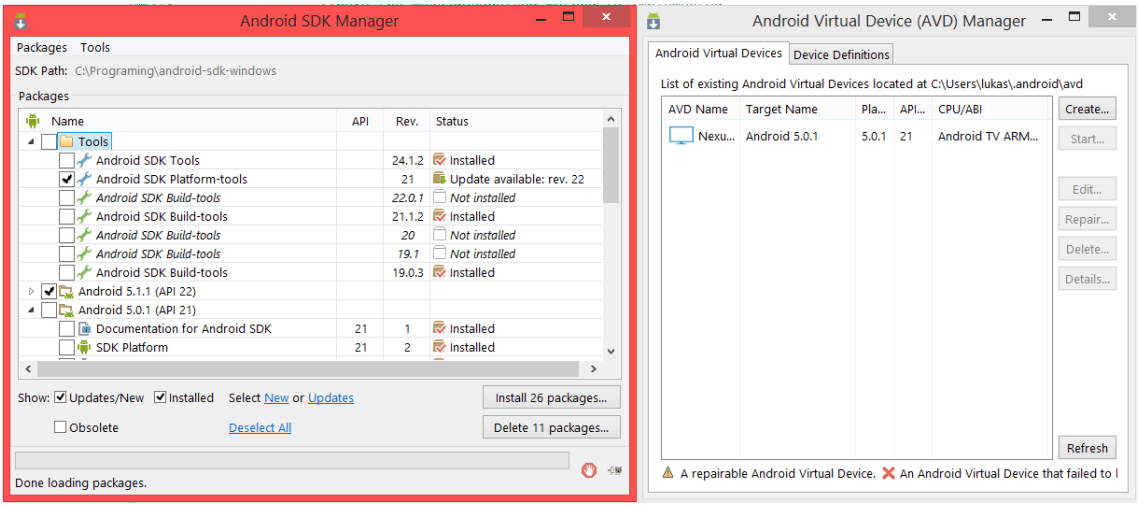
Vývoj aplikácií je možný na všetkých hlavných platformách – Linux, Mac a Windows. Hovoríme o Host-Target vývojovom prostredí, pretože prostredie, v ktorom je aplikácia vyvíjaná, a prostredie, v ktorom sa používa, sú odlišné.
Pri vývoji aplikácií a na testovanie ich funkcionality je potrebné nainštalovať viacero komponentov. Tie sú však bezplatné. [1]

3.2.1 Java Development Kit (JDK)

Vyvíjaná aplikácia bude napísaná v jazyku Java, preto treba nainštalovať základné knižnice, applety a nástroje pre platformu Java obsahujúce JDK. Možno použiť aj knižnice C a C++. Nájdu sa pod názvom Native Development Kit (NDK). Tie je vhodné použiť pre aplikácie s rýchlou odozvou (prevažne hry). S ohľadom na náročnosť aplikácie na riadenie robota je postačujúce použiť JDK.
Základnými časťami JDK sú Java Runtime Enviroment (JRE) slúžiaci na spustenie aplikácií vývojových nástrojov, ďalej prekladač, debugger atď.
Aktuálna verzia sa nachádza na stránkach výrobcu:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

3.2.2 Software Development Kit (SDK)

Softvér obsahuje aplikáciu Android SDK a AVD Manager. Cez túto aplikáciu možno stiahnuť jednotlivé komponenty SDK – rôzne Android SDK. Obsahuje emulátor Android zariadení. Emulátor je samostatná desktopová aplikácia umožňujúca testovať aplikácie bez mobilného zariadenia. Pomocou Android SDK AVD Managera možno vytvárať a spúšťať Android Virtual Device – nakonfigurované inštalácie emulátora. Umožňuje veľmi presne vymodelovať vlastnosti konkrétneho zariadenia. [3]



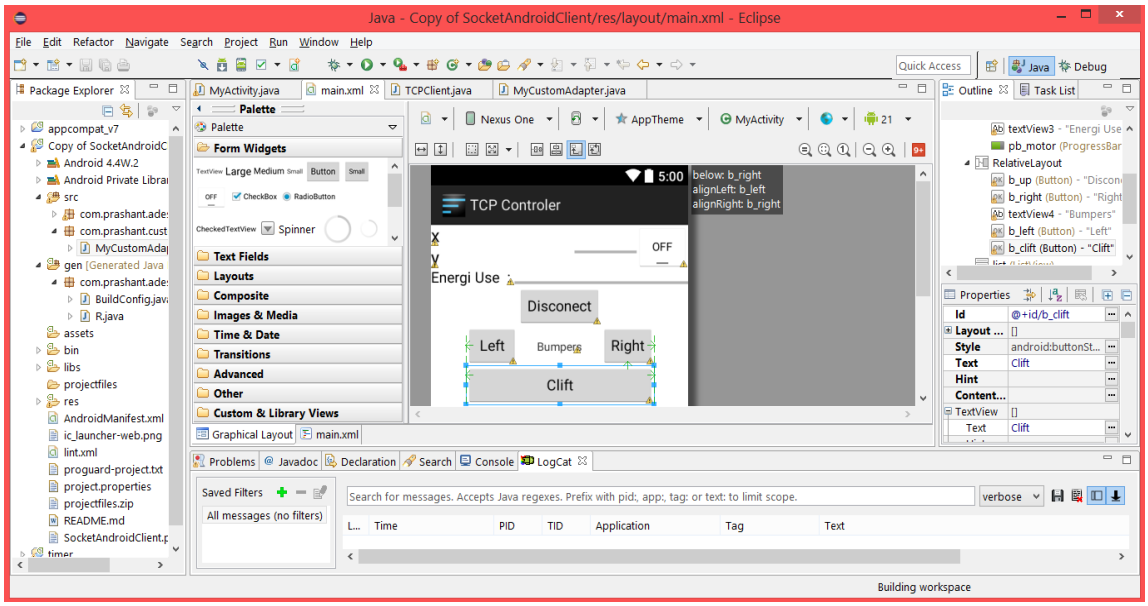
Obrázok 4: Android SDK Manager a AVD Manager

Najdôležitejšími komponentmi v SDK sú platformy Android. Tie zodpovedajú produkčným platformám bežiacim na skutočných zariadeniach. Každá platforma obsahuje systémové knižnice, systémový obraz, vizuálne prostredie emulátora, ukážky kódu a iné zdroje špecifické pre platformu. [3]
Aktuálnu verziu je možné stiahnuť na stránkach vývojárov Android:
https://developer.android.com/sdk/installing/index.html

3.2.3 Vývojové prostredie

Na tvorbu aplikácií pre Android postačí textový editor alebo akékoľvek prostredie umožňujúce programovať v jazyku Java, ako je Netbeans, Oracle jDeveloper, BlueJ. Pre programátora je však pohodlnejšie využiť nástroj prispôsobený na tvorbu Android aplikácií. Takýchto nástrojov je veľké množstvo, napríklad DroidEditor, JavaIDEdroid, Adroid Java Editor, C4droid a Eclipse, ktorý bol využitý na vývoj aplikácie.

- **Eclipse** – je donedávna oficiálne podporované open source vývojové prostredie pre Android aplikácie. Po stiahnutí Eclipse treba doinštalovať plugin ADT (Android Development Tools). Ten pridá funkcionalitu na vývoj Android aplikácií. Jeho vývoj je už ukončený.



Obrázok 5: Prostredie Eclipse

- **Android Studio** – je prvé oficiálne vývojové prostredie pre Android. Android Studio získalo všetky výhody nástroja IntelliJ IDEA, ako sú analýza kódu, automatické dokončovanie alebo refaktoring. Umožňuje aj jednoduchú internacionalizáciu. Na vývoj lepšieho a efektívnejšieho kódu je k dispozícii monitor pamäti. Výhodou je tiež možnosť udržiavať viac paralelných verzií výslednej aplikácie, napríklad voľne šíriteľné a platené verzie. Medzi ďalšie podstatné vlastnosti patrí prístup ku cloudovým službám Google. Android Studio možno od júna 2013 stiahnuť bezplatne. [4]

3.3 Základné časti aplikácie Android

Aplikácie Android sa skladajú zo štyroch základných typov komponentov:

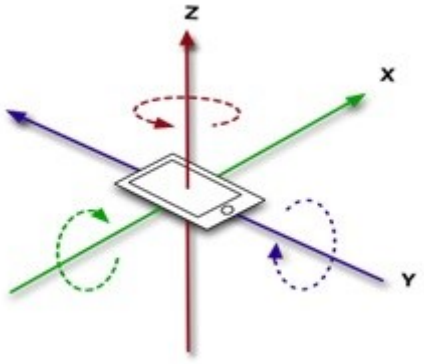
- **activities –** komponenty užívateľského rozhrania, ktoré zodpovedajú jednej obrazovke. Obsahujú grafické, užívateľské rozhranie na interakciu s užívateľom. Aplikácie obvykle obsahujú väčšie množstvo aktivít, medzi ktorými užívateľ môže prepínať. Aktivita sa môže nachádzať v štyroch stavoch:
    - activity starts – začiatok, po inicializácii aktivity,
    - activity is running – aktivita je zobrazená a je možná interakcia s užívateľom; len jedna aktivita sa môže nachádzať v tomto stave,
    - process is killed – aktivita je zrušená pre nedostatok pamäte,
    - activity is shut down – aktivita je zrušená;
- **servicies** – procesy bežiace na pozadí, využívajú sa na vykonávanie dlhotrvajúcich procesov, napríklad pripojenie k serveru;
- **content providers** – aplikačné rozhranie na zdieľanie dát medzi aplikáciami, ale aj medzi jednotlivými aktivitami;
- **broadcast receivers** – komponent slúžiaci na „počúvanie“ oznámení a reakciu na tieto udalosti, napríklad zobrazenie notifikácie o prijatej správe. [1]

3.4 Použitie triedy Sensors

Na získanie dát zo senzorov sa použije trieda Sensors. Sprostredkováva dáta zo všetkých senzorov, ktoré MT obsahuje. Senzory sa delia do troch základných skupín:

- motion sensor – poskytuje informácie o veľkosti zrýchlenia a rotačných rýchlostí v troch osiach; do tejto kategórie patria akcelerometer a gyroskop;
- enviromental sensors – poskytujú informácie o okolí MT, ako sú teplota okultného vzduchu, tlak, vlhkosť a intenzita okultného osvetlenia;
- position sensors – poskytujú informácie o fyzickej polohe zariadenia; skupina obsahuje orientačný senzor a magnetometer.

S ohľadom na funkcionalitu aplikácie na riadenie je vhodné použiť skupinu position sensors a vo finálnej aplikácii sú použité orientačné senzory, ktoré poskytujú informácie v troch osiach.



Obrázok 6: Zobrazenie osí pozičných senzorov

3.4.1 SensorManager a SensorEventListener

SensorManager poskytuje prístup k senzorom za predpokladu, že zariadenie je vybavené týmito senzormi. Sú použité API, ktoré podporujú využitie zvoleného druhu senzorov.
SensorEventListener zabezpečuje vykonávanie kódu v tom prípade, že získa údaje zo senzorov.
V nasledujúcej časti je opísaná implementácia v jednotlivých častiach Android aplikácie.

3.4.2 Štruktúra aplikácie

Aplikácia je tvorená hlavnou triedou `MainActivity` rozširujúcou triedu `Activity`. Táto trieda obsahuje metódy, ktoré určujú správanie v jednotlivých životných cykloch aktivity:

- **onCreate** – v tejto časti sa implementuje správanie, ktoré sa vykoná po spustení hlavnej triedi `MainActivity`,
- **onPause, onResume** – dvojica metód, ktorá umožňuje upraviť správanie pri dočasnom zastavení a opätovnom spustení aktivity,
- **onSensorChange** – metóda vykonávaná v prípade, že dôjde k zmenám hodnôt sledovaných senzorov.

Registrácia

V hlavnej triede sa registruje trieda `SensorManager` a `Sensor`. Premenná `ManagerDelay` predstavuje hodnotu oneskorenia pred získaním novej hodnoty od senzorov a môže nadobúdať nasledujúce hodnoty:

- SENSOR_DELAY_FASTEST,
- SENSOR_DELAY_GAME,
- SENSOR_DELAY_NORMAL,
- SENSOR_DELAY_UI.

```
public class MainActivity extends Activity implements SensorEventListener {
	private  SensorManager mSensorManager;
	private  Sensor mOrientation;
	private int ManagerDelay = SensorManager. SENSOR_DELAY_NORMAL;
...
```

Po spustení aktivity

V metóde `onCreate` sa vytvára SensorManager a prideľuje sa mu Listener. Nastavuje sa druh senzora, metódou `getDefaultSensor` s parametrom `Sensor.TYPE_ORIENTATION` a ďalej sa určuje, ako často z neho budú získavané údaje; toto nastavenie je uložené v `ManagerDelay`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	//create orientation Sensor
	this.mSensorManager =(SensorManager)getSystemService(SENSOR_SERVICE);
	this.mOrientation = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
	this.mSensorManager.registerListener(this,this.mOrientation,ManagerDelay);
```

Po zmene hodnôt senzora

`onSensorChanged` je metóda Listeneru vykonávaná po získaní dát zo senzorov. V tejto metóde možno spracovávať získané hodnoty, ktoré sa nachádzajú v `arg0.values.`
Tieto hodnoty sú z nasledovných intervalov:

- X < 0 ; 360 >
- Y < -180 ; 180 >
- Z < -180 ; 180 >

```
public void onSensorChanged(SensorEvent arg0) {
        float [] orientation = arg0.values;
        String x =" x: " +orientation[0];
        String y =" y: " +orientation[1];
        String z =" z: " +orientation[2];
        Textwiev1.setText(x + z+ y);
}
```

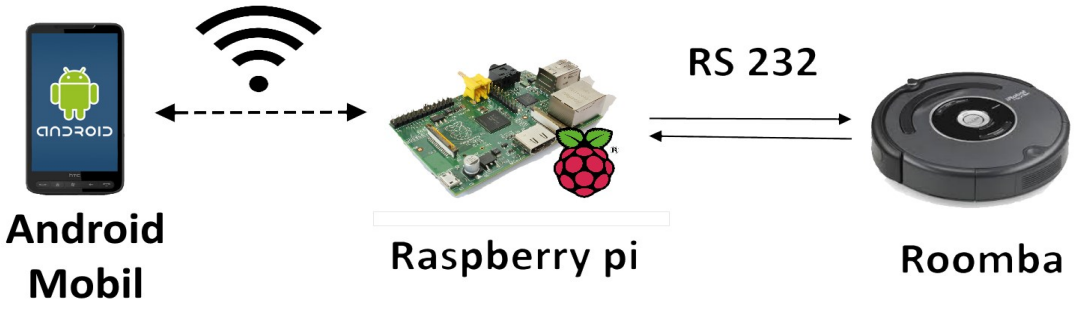
Správanie v prípade pozastavenia a prebudenia
V prípade pozastavenia aplikácie je potrebné odregistrovať Listener v metóde `onCreate` na korektné ukončenie aplikácie. V prípade opätovného spustenia aplikácie treba tento Listener znovu registrovať.

```
protected void onResume(){
        mSensorManager.registerListener(this, mOrientation, ManagerDelay);
        super.onResume();
}

protected void onPause(){
        super.onPause();
        mSensorManager.unregisterListener(this);
}
```

4 Návrh obojsmerného komunikačného rozhrania na prenos dát z MT do riadiaceho počítača robotického podvozka

Ako podvozok sa využíva robotický vysávač iRobot Roomba, s ktorým je možná komunikácia dvomi spôsobmi. Prvým je pridanie Wi-Fi modulu, ktorý pod označením RooWiFi distribuuje výrobca iRobot Roomba. Tento modul poskytuje komunikáciu prostredníctvom TCP/IP socketu a podporuje vývoj aplikácií v jazykoch Ajax, JSON, XML, C, Python, Java a ďalších. Druhou možnosťou je komunikácia prostredníctvom sériovej linky. Špecifikovaná je bližšie v štandarde RS-232. Z dôvodu možného prerušenia Wi-Fi prepojenia zapríčiňujúceho stratu kontroly nad RPR je zvolená komunikácia prostredníctvom RS-232. Tá poskytuje trvalú komunikáciu s RPR. Na sprostredkovanie komunikácie medzi RPR a MT je použité Raspberry Pi. Zabezpečuje komunikáciu medzi mobilným telefónom s operačným systémom Android prostredníctvom Wi-Fi a komunikáciu s RPR prostredníctvom RS-232.



Obrázok 7: Komunikačné rozhranie

4.1 Hardvérové komponenty komunikačného rozhrania

Na sprostredkovanie komunikácie je nutné použiť viacero hardvérových komponentov. Tie sú opísané v nasledujúcich kapitolách.

4.1.1 Mobilný telefón

V tejto práci sa používa mobilný telefón s operačným systémom Android. Aplikácia na riadenie je napísaná pre tento operačný systém. Je však možné tento MT nahradiť zariadením s iOS od spoločnosti Apple, mobilným telefónom s OS Windows alebo akýmkoľvek zariadením. Toto zariadenie podporuje komunikáciu prostredníctvom WI-FI a má polohové senzory. V prípade nahradenia MT Android je však nutné aplikáciu previesť na požadovanú platformu.

Požiadavky na hardvér mobilného zariadenia:

- minimálne hardvérové požiadavky na chod Android API 8 alebo vyššie,
- WI-FI modul,
- pozičné senzory.

4.1.2 Raspberry Pi

Raspberry Pi je jednodoskový počítač s doskou veľkosti kreditnej karty. Vyvíja ho britská nadácia Raspberry Pi Foundation s cieľom podporiť výučbu informatiky na školách. Základom je jednočipový procesor BCM2835 firmy Broadcom, ktorý obsahuje centrálny procesor ARM1176JZF-S s taktom 700 MHz. Ďalej je to grafický procesor VideoCore IV a 256 MiB (model A) alebo 512 MiB (model B) s pamäťou RWM (RAM). Neobsahuje žiadne rozhranie na pevný disk či SSD na zavádzanie systému. Na trvalé uschovanie dát je určený slot na SD kartu. [5]

Hardvérové parametre Raspberry Pi 1 Model B

- Procesor BCM2835 z rodiny ARM Cortex-A6 taktovaný na 700 MHz.
- Grafický procesor VideoCore IV podporujúci OpenGL ES 2.0, 1080p30, MPEG-4.
- Obrazový výstup Composite RCA, HDMI, DSI.
- Zvukový výstup cez 3,5 mm konektor, HDMI.
- 12xGPIO, UART, I²C, zbernica SPI.
- Podpora JTAG Debug.
- Watchdog timer.
- 512 MiB RWM (RAM) zdieľaný s grafickou kartou.
- Slot na micro SD kartu.
- Dva USB porty.
- Eternetový adaptér 10/100 s konektorom RJ45.

WI-FI Modul

Keďže Raspberry Pi neobsahuje WI-FI modul, je nutné použiť USB WI-FI adaptér. Napríklad ELEMENT14 WIPI Frequency RF:2.4GHz. Ten podporuje štandardy IEEE 802.11b/g.

RS-232 prevodník

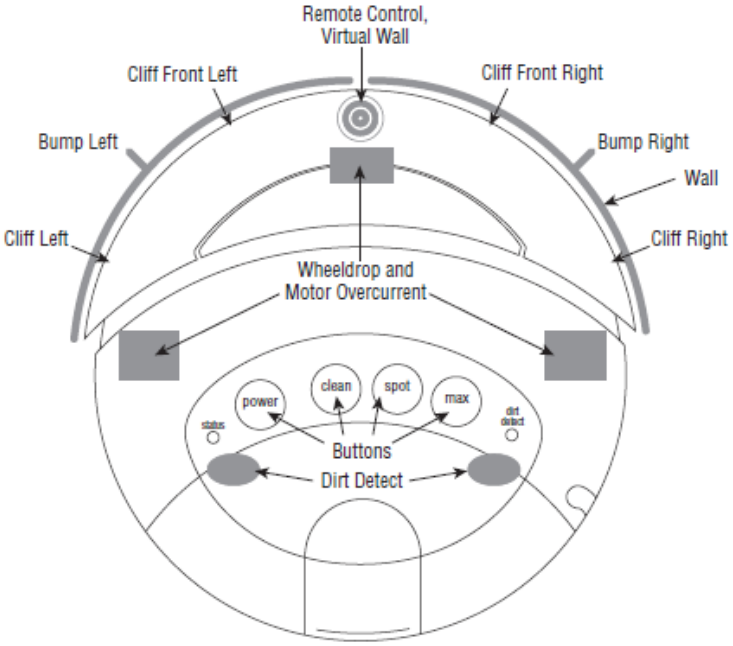
Raspberry Pi podporuje komunikáciu prostredníctvom sériovej komunikácie. Ďalej podporuje zapojenie pomocou RX, TX a GND. Tie sú na využitie na komunikáciu s RPR nedostačujúce. Preto je potrebné využiť výstup RTS pre budiaci signál RPR. Treba použiť RS-232 USB prevodník, napríklad CP2012 Master Chip Standard USB 2.0.

DC/DC prevodník

Z dôvodu mobility je potrebné riešiť napájanie Raspberry Pi batériou. V tomto riešení je použitá 12V 4,5Ah batéria a nastaviteľný napäťový menič prevádzajúci 12 V na 5 V, ktoré požaduje Raspberry Pi.

4.1.3 iRobot Roomba

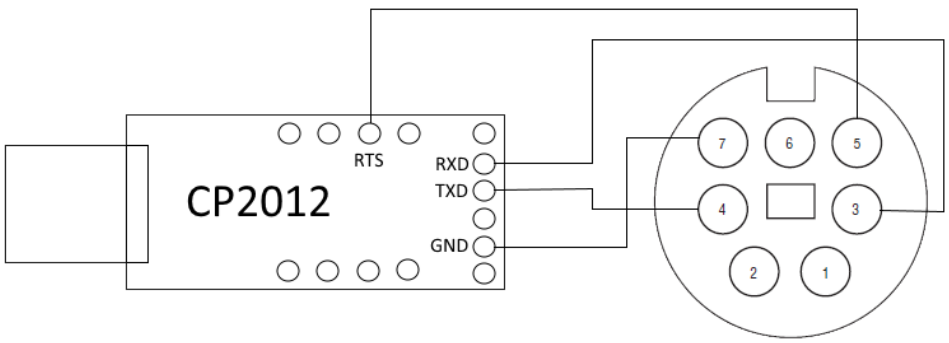
Roomba je robotický vysávač od spoločnosti iRobot. Ide o plne automatického robota, s ktorým je možná komunikácia prostredníctvom sériového rozhrania. Jeho pohyb zabezpečujú dve kolesá s vlastnými motormi. Tie sú napájané zo zabudovanej batérie. Robot je vybavený reproduktorom a svetelnou signalizáciou. Na pohyb a vysávanie využíva 38 senzorov určujúcich informácie o okolí a stavoch zariadenia. Na aplikáciu z týchto senzorov za účelom riadenia pohybu sa využívajú senzory nárazu a senzory útesu. Ďalšie užitočné informácie sa získavajú z interného stavu robota, ako sú stav batérie a aktuálna spotreba energie. [1]



Obrázok 8: Rozloženie senzorov RPR [1]

Fyzické prepojenie

Na komunikáciu s Roombou je použité sériové rozhranie. Prepojenie Raspberry Pi a Roomby sa realizuje pripojením pomocou Mini-DIN konektora a RS-232 USB prevodníka. Prepojenie je realizované podľa nasledujúceho obrázka.



Obrázok 9: Schematické zapojenie CP2012 a Mini-DIN konektora

4.2 Komunikácia medzi Raspberry Pi a robotickým podvozkom iRobot Roomba

Ako už bolo uvedené, RPR komunikuje prostredníctvom sériovej linky. Špecifikácie komunikácie sa nachádzajú v iRobot Roomba Serial Command Interface (SCI) [13]. Tie sú voľne stiahnuteľné zo stránok spoločnosti iRobot. Podrobnejšie informácie sa nachádzajú v publikácii Hacking Roomba, autor Tod E. Kurt. [2]

Na komunikáciu Raspberry Pi s RPR sa používa mnou vytvorená trieda Roomba SCI, ktorá je naprogramovaná v jazyku C++. Táto trieda umožňuje vytvorenie sériovej komunikácie, spustenie Roomby, riadenie pohybu, ovládanie svetelnej a zvukovej signalizácie. Umožňuje získať informácie aj zo senzorov. V nasledujúcich kapitolách budú bližšie opísané jednotlivé časti Roomba SCI a správanie RPR.

4.2.1 Nastavenie sériového portu

Špecifikácie sériového portu Roomba:

- Baud: 57 600 baud alebo 19 200 baud.
- Data bits: 8.
- Parity: None.
- Stop bits: 1.
- Flow control: None.

Band určuje rýchlosť prenosu po sériovej linke. Tá je štandardne nastavená na 57 600 baud. Túto hodnotu možno manuálne zmeniť na 19 200 baud alebo ju nastaviť pomocou sériového rozhrania v dvanástich stupňoch od 300 baud po 115 200 baud. [1]

```
RoombaSCI::RoombaSCI(){

    sercmd = TIOCM_RTS;

    fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY); //otvorenie sériového portu
    if (fd == -1)
    {
        printf( "-!- Error opening port /dev/ttyUSB0\n");
    }
    tcgetattr(fd, &options);                    //získanie aktuálnych nastavení portu
    cfsetispeed(&options, B57600);              //nastavenie bound pre vstup
    cfsetospeed(&options, B57600);              //nastavenie bound pre výstup
    cfsetispeed(&options, CS8);                 //nastavenie Data bits pre vstup
    cfsetospeed(&options, CS8);                 //nastavenie Data bits pre výstup
    options.c_cflag |= (CLOCAL | CREAD);        //Enable the receiver and set local mode
    options.c_cflag &= ~PARENB;                 //Nastaveni Mask Parity na No Parity
    options.c_cflag &= ~CSTOPB;                 //Nastaveni Stop Bits na 1
    options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); // výber surového vstupu
    options.c_iflag |= (INPCK | ISTRIP);        // nastavenie vstupu na No Parryty
    options.c_iflag &= ~(IXON | IXOFF | IXANY); // nastavenie Flow control na None

    tcsetattr(fd, TCSANOW, &options); //Nastavenie portu podľa nových nastavení
    printf( "Create SCI  \n ");
}
```

Výpis kódu 1: Vytvorenie a nastavenie sériového portu v konštruktore triedy Roomba SCI

4.2.2 Spôsob komunikácie

Komunikácia prebieha pomocou operačných kódov, ktoré sú v 8-bitovej forme zasielané RPR. Vybrané kódy použité v Roomba SCI sú uvedené v tabuľke 1. Tieto kódy možno použiť len v mode pre nich určenom. Každá Roomba má tri druhy modov. Tie udeľujú oprávnenia na vykonávanie príkazov.

Druhy modov

- **Passive Mode** – je možné používať príkazy ako vysávanie, zaparkovanie do nabíjacej stanice, spustenie dema a získať informácie zo senzorov.
- **Safe Mode** – sú sprístupnené všetky príkazy s výnimkou podmienok súvisiacich s bezpečnosťou, ako sú detekcia útesu, detekcia spadnutých kolies, pripojenie na nabíjačku.
- **Full Mode** – sú sprístupnené všetky povely bez ohľadu na bezpečnosť. [1]

Príkaz	Operačnýkód	Počet dát Byt	Význam	Použitie v mode
Start	128	0	Aktivácia Roomba I/O rozhrania	All
Safe	131	0	Nastavenie Safe modu	All
Power	133	0	Vypnutie Roomba	All
Drive	137	4	Príkaz na pohyb Roomby	Safe
LEDs	139	3	Nastavenie svetelnej signalizácie	Safe
Song	140	4+2x	Uloženie hudobnej sekvencie	Safe
Play	141	1	Prehranie hudobnej sekvencie	Safe
Sensor	142	1	Vyžiadanie dát zo senzorov	All

Tabuľka 1: Operačné príkazy Roomba SCI

Aby bolo možné aktivovať I/O rozhranie pomocou príkazu Start, je nutné, aby bola Roomba v Stand By režime. To je možné dvomi spôsobmi: 1. použiť tlačidlo napájania a 2. použiť budiacu sekvenciu nastavením RTS.

```
bool RoombaSCI::StartSafe(){
     //Zobudenie pomocou RTS
    ioctl(fd, TIOCMBIC, &sercmd);   // Reset RTS pin.
    usleep(10000);                  // Presné určené čakanie pri budiacej sekencii
    ioctl(fd, TIOCMBIS, &sercmd);   // Reset RTS pin.
    ioctl(fd, TIOCMGET, &serstat);  // Set RTS pin.
    sleep(2);                       // Presné určené čakanie pri budiacej sekencii

    char start = 0x80 ;               // 128
    char control=0x82;                // 130
    write(fd,&start,1);               // Odoslanie operačného príkazu
    usleep(20000);                    // Nutné čakanie pri prepínaní modov
    write(fd,&control,1);             // Odoslanie operačného príkazu
    usleep(20000);                    // Nutné čakanie pri prepínaní modov
    return true;
}
```

Výpis kódu 2: Kód na spustenie Safe modu

4.2.3 Riadenie pohybu

Na riadenie pohybu sa využíva funkcia Drive s parametrami Velocity a Radius. Táto funkcia odosiela sekvenciu piatich bajtov. Prvý bajt predstavuje operačný kód a v nasledujúcich dvoch bajtoch je uložená hodnota Velocity. Tá určuje rýchlosť pohybu dopredu alebo dozadu. Posledné dva bajty slúžia na hodnotu Radius určujúcu polomer kružnice, po ktorej sa Roomba pohybuje.

Možné hodnoty parametrov:
Velocity – od -500 do 500 mm/s
Radius – od -2 000 do 2 000 mm

Špeciálny význam hodnôt:

- **32 768** – pohyb rovno
- **1** – otáčanie na mieste proti smeru hodinových ručičiek
- **-1** – otáčanie na mieste v smere hodinových ručičiek

Obrázok 10: Vizuálne zobrazenie pohybu pri niektorých hodnotách rádiusu

Poznámka 4.2.3

Pretože parametre Velocity a Radius sa posielajú ako dvojbajtová hodnota, možno parameter rozdeliť na dva bajty nasledovným spôsobom. Z hodnoty Velocity sa uloží do premennej `hig_velocity` 8 bajtov s najvyššou prioritou a do hodnoty `low_velocity` zvyšných 8 bajtov s nižšou prioritou.
Po rozdelení hodnôt Velocity a Radius sa posiela nasledujúca sekvencia príkazov Roombe pomocou sériovej linky [Operačný kód (137)] [high velocity] [low velocity] [high radius] [low radius] .

```
char high_velocity = char( (Velocity >> 8) & 0xff);
char low_velocity = char(Velocity & 0xff);
```

4.2.4 Získavanie hodnôt zo senzorov

Prevádza sa funkciou Sensor. Táto funkcia odošle operačný kód Sensor a hodnotu, ktorá určuje skupinu senzorov. V tejto implementácii sa získavajú dáta zo všetkých senzorov, ale spracované sú len hodnoty nachádzajúce sa v nasledujúcej tabuľke.

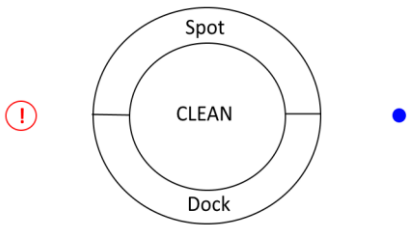
Názov	Možné hodnoty	Význam
WheeldropCenter	True / False	Detekcia pádu kolesa
WheeldropRight	True / False	Detekcia pádu kolesa
WheeldropLeft	True / False	Detekcia pádu kolesa
BumpLeft	True / False	Detekcia nárazu
BumpRight	True / False	Detekcia nárazu
CliffLeft	True / False	Detekcia útesu
CliffFrontLeft	True / False	Detekcia útesu
CliftRight	True / False	Detekcia útesu
CliftFrontRight	True / False	Detekcia útesu
Clift	True / False	Detekcia útesu (ktorýkoľvek senzor)
Capacity	0 – 65 535	Odhadovaná kapacita batérie v mAH
Charge	0 – 65 535	Aktuálny stav batérie v mAH
Current	-32 768 – 32 767	Aktuálny prúd v mA, záporná hodnota označuje spotrebu a kladná nabíjanie
Voltage	0 – 65 535	Napätie batérie v mV

Tabuľka 2: Prehľad senzorov

4.2.5 Ostatné príkazy

Power – slúži na vypnutie Roomby. Po tomto vypnutí je nutné Roombu prebudiť manuálne stlačením tlačidla napájania alebo štartovacou sekvenciou prostredníctvom sériovej linky.

Leds – umožňuje aktiváciu svetelnej signalizácie BlueDiod, RedWarning, GreanSpot, GreanDock, nastavenie časti CLEAN pomocou hodnôt PowerColor a PowerIntensity. Hodnota PowerColor určuje nastavenie farby, ktoré sa nastavuje od 0 (zelená) do 255 (červená). Hodnota PowerIntensity určuje intenzitu od 0 (vypnuté) do 255 (maximálna intenzita).



Obrázok 11: Svetelná signalizácia

Song – umožňuje nahrať hudobnú sekvenciu. Zadáva sa dvojicou hodnôt, kde prvá hodnota predstavuje frekvenciu a druhá čas prehrávania tejto frekvencie. Je nutné zadať číslo hudobnej sekvencie a dĺžku, pričom dĺžka je v rozmedzí 1 – 16 záznamov.

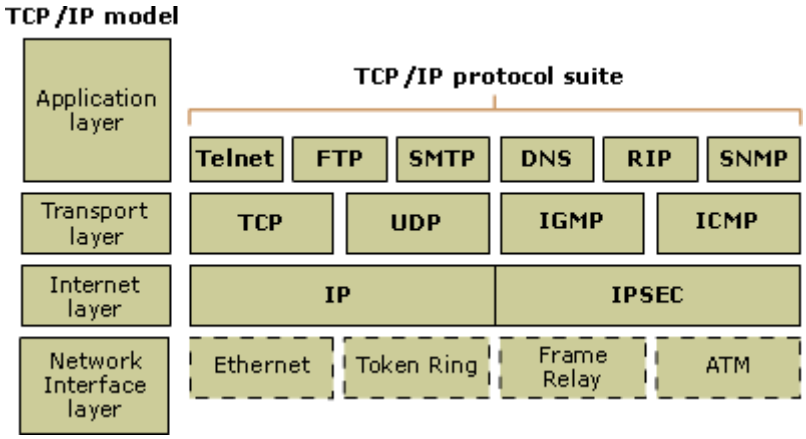
Play – slúži na prehrávanie uložených hudobných sekvencií. Funkcia odosiela riadiaci kód a číslo hudobnej frekvencie, ktorá je uložená pomocou metódy Song.

4.3 Komunikácia medzi mobilným telefónom a Raspberry Pi

Realizovaná je pomocou bezdrôtovej komunikácie Wi-Fi, ktorej vlastnosti sú dané v štandarde IEEE 802.11x (x označuje jednotlivé štandardy). Na náš cieľ postačuje zabezpečiť, aby všetky zariadenia používané na vytvorenie bezdrôtovej komunikácie podporovali rovnaké štandardy.
Bezdrôtovú komunikáciu možno realizovať dvomi spôsobmi. Prvý je pripojenie oboch zariadení do existujúcej Wi-Fi siete. Druhý je realizovateľný za predpokladu, že mobilný telefón, prípadne Raspberry Pi podporujú vytvorenie prístupového bodu Wi-Fi. Tento spôsob umožní vytvorenie mobilnej Wi-Fi siete.
Samotná komunikácia medzi klientom nachádzajúcim sa na MT a serverom, ktorý je umiestnený na Raspberry Pi, je realizovaná pomocou Socket serveru. Ten komunikuje pomocou protokolu TCP/IP.

4.3.1 Model protokolu TCP/IP

Tento model je založený na štvorvrstvovom referenčnom modeli. Všetky protokoly nachádzajúce sa v sade protokolov TCP/IP patria do troch vrchných vrstiev tohto modelu.



Obrázok 12: Poskytované služby a protokoly vo vrstvách modelu TCP/IP [7]

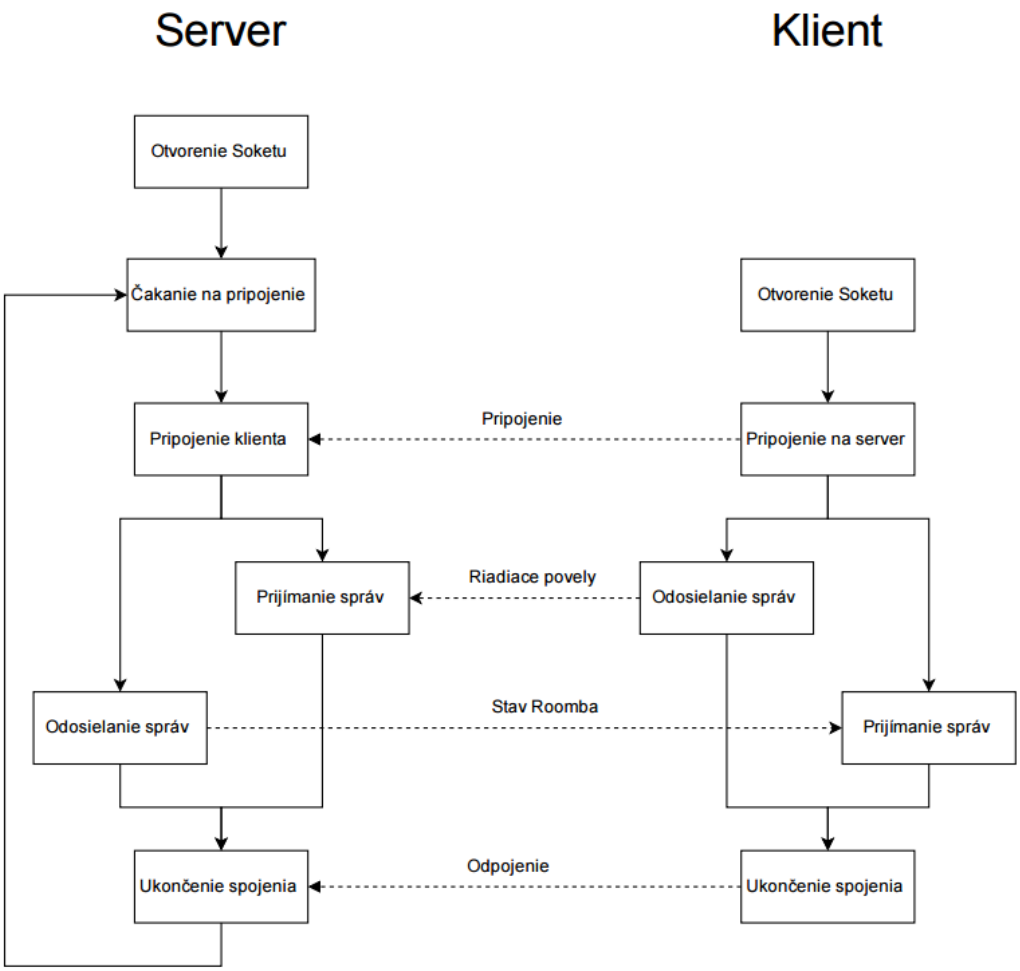
- **Aplikačná vrstva** – definuje aplikačný protokol TCP/IP a spôsob spolupráce s klientskym programov so službami. V tejto časti sa využíva socket.
- **Transportná vrstva** – zabezpečuje správu komunikačných relácií medzi hostiteľskými počítačmi. Definuje úroveň služieb a stav pripojenia pri prenose dát.
- **Vrstva internet** – vkladá dáta do balíčka IP obsahujúceho informácie o zdrojovej a cieľovej adrese a stará sa o smerovanie.
- **Sieťové rozhranie** – určuje podrobnosti tykajúce sa fyzického prenosu dát po sieti vrátane ich prevodu na elektrické signály používané hardvérovými zariadeniami. [7]

4.3.2 Komunikácia medzi serverom a klientom

S ohľadom na bezpečnosť a plynulosť riadenia RPR sa získavajú dáta zo senzorov častejšie ako sa spracúvajú riadiace povely. Pre rýchlosť odozvy celého systému je upravený základný komunikačný model Socket serveru pochádzajúci od Rob Tougher, ktorý ho ako Open Publication License poskytuje pod názvom Linux Socket Programing In C++. [8]

Opis komunikácie

Na strane serveru je vytvorený socket, kde sa čaká, kým nepríde požiadavka na pripojenie zo strany klienta. Keď server túto požiadavku prijme, vytvorí sa pripojenie, ktoré je vytvorené aj na strane klienta. Riadiaca aj stavová komunikácia na strane klienta aj na strane serveru sa realizuje paralelne a prebieha, kým sa klient neodpojí. Po odpojení klienta dochádza k ukončeniu komunikácie a ukončeniu vlákien na prijímanie; odosielanie správ sa následne na server opätovne dostáva do stavu čakania na pripojenie.



Obrázok 13: Komunikácia medzi serverom a klientom

4.3.3 Realizácia Socket Server v C++

Ako už bolo uvedené v kapitole 4.3.2 *Komunikácia medzi serverom a klientom,* na vytvorenie serveru sa používa upravená implementácia Socket serveru od Rob Taugher. Základná implementácia využíva viaceré knižnice, ktoré nebolo potrebné upravovať.

Použité knižnice:

- **Socket.h, Socket.cpp** – trieda, ktorá implementuje potrebné funkcie v Socket API.
- **SocketException.h** – knižnica výnimiek, ktoré môžu nastať pri komunikácii pomocou socketu.
- **ServerSocket.h, ServerSocket.cpp** – trieda obsahujúca konštruktor vytvárajúci spojenie, metódu na pripojenie klienta, metódu na prijatie a odoslanie správ.

Základná implementácia Socket Serveru

Princíp funkcie základnej implementácie Socket Serveru spočíva v tom, že na začiatku sa pokúsi aplikácia vytvoriť `ServerSocket` na sockete 8888. Ak tento port je dostupný a nenastane výnimka, aplikácia vytvorí nový socket a čaká na pripojenie zo strany klienta. Ak sa klient pripojí, aplikácia v cykle prijíma a vracia dáta, kým sa klient neodpojí. V tom prípade ukončí spojenie a vytvorí nový socket a čaká na pripojenie klienta.

```
int main ( int argc, int argv[] )
{
  try
    {
      ServerSocket server ( 8888 );          //vytvorenie serveru Sockete 8888
        while ( true )
        {
          ServerSocket new_sock;
          server.accept ( new_sock );        //vytvorenie nového pripojenia
          try
            {
                while ( true )
                {
                  std::string data;
                  new_sock >> data;          //prijatie správy
                  new_sock << data;          //zaslanie správy
                }
            }
          catch ( SocketException& ) {}
        }
    }
  catch ( SocketException& e )
    {
      std::cout << "Exception was caught:" << e.description() << "\nExiting.\n";
    }
  return 0;
}
```

Výpis kódu 3: Základná implementácia Socket Serveru

Rozšírenie implementácie Socket Server

Z vyššie uvedených dôvodov je použitá paralelná implementácia prijímania na odosielania dát. Na implementáciu sa používa knižnica `pthread.h`, ktorá umožňuje implementáciu vláken.

Na odovzdávanie určitých parametrov vláknu bola vytvorená štruktúra `Argument`, ktorá obsahuje premenné. Hodnota `delay` určuje čas oneskorenia pred ďalším vykonávaním cyklu. V kapitole 5 bude bližšie vysvetlený jej význam.

```
struct Argument{
    int delay;
};
```

Na implementáciu vlákna je vytvorená trieda vlákno, v ktorej je demonštrované použitie štruktúry argument. Z nej preberáme hodnotu `delay`. Táto hodnota určuje čas (v sekundách) uspania nekonečného cyklu while potom, čo bol odoslaný text `"niake data\n"` klientovi.

```
void * vlakno(void *parameters) {
    struct Argument *p = (struct Argument *) parameters;
    int hodnota = p->delay;
    while (1) {
        new_sock <<  "niake data\n";
        sleep(hodnota);
    }
}
```

V poslednom bloku kódu je vytvorené vlákno metódou `pthread_create` v časti, kde sa vytvára nové spojenie po pripojení klienta.

```
  try
    {
      ServerSocket server ( 8888 );
    while ( true )
    {
      pthread_t thr;
      struct Argument a;
      a.delay =100000;

      server.accept ( new_sock );
      if (pthread_create(&thr, NULL, &vlakno, &a)) { puts("Chyba vlákna");return 1;}
     …
```

Výpis kódu 4: Rozšírenie základnej implementácie Socket Serveru

4.3.4 Realizácia Socket Client v Android

Komunikácia zo serverom je realizovaná pomocou triedy `TCPClient.java`, ktorá je prebraná z návodu Android TCP Connection Tutorial. Táto trieda zabezpečuje všetku komunikáciu zo serverom, ktorý bol opísaný v predchádzajúcich kapitolách. Trieda obsahuje tri pôvodné a jednu pridanú funkciu, ktoré sa používajú na komunikáciu: [9]

- **run()** – trieda run vytvára socket pre sieťovú komunikáciu. Po nadviazaní komunikácie funkcia obsahuje cyklus, ktorý sa stará o načítanie prichádzajúcich správ zo serveru. Tento cyklus prebieha dovtedy, kým nie je zastavený.
- **destroyClient()** – jedna z funkcií na zastavenie a vykonávanie cyklu vo funkcii run(). Okrem zastavenia prijímania dát zo serveru ukončuje spojenie so serverom a zatvára socket.
- **sendMessage (String message)** – funkcia slúžiaca na odosielanie správ, ktoré sú odovzdané ako reťazec String.
- **Construct()** – vytvorená trieda zastupujúca metódy, ktoré obsahoval pôvodný konštruktor triedy `TCPClient.java`. Dôvod vytvorenia tejto funkcie bude spomenutý v kapitole 6 o testovaní odozvy a spoľahlivosti systému.

Okrem vyššie uvedených metód, ktoré obsahuje trieda `TCPClient.java` je potrebná implementácia interface metódy `OnMessageReceived(String message)`, a to v `MyActicvity`. Táto metóda je vykonávaná asynchrónne na pozadí a volá metódu `onProgressUpdate`. Tá sa stará o spracovanie prijatých správ dewe.

```
public class connectTask extends AsyncTask<String,String,TCPClient> {
    @Override
    protected TCPClient doInBackground(String... message)
    {
        //vytvorenie TCPClient objektu
           mTcpClient.Construct(new TCPClient.OnMessageReceived()
        {
            @Override
            //implementácia messageReceived
            public void messageReceived(String message)
            {
                  try
                  {
                         //Metóda volajúca OnProgress Update
                         publishProgress(message);
                  }
                  catch (Exception e)
                  {
                         e.printStackTrace();
                  }
            }
    });
    ...
```

Výpis kódu 5: OnMessageReceived

4.3.5 Komunikačný protokol

Komunikácia medzi serverom a klientom je realizovaná zasielaním nešifrovaných textových reťazcov, ktorých tvar je určený v dvoch protokoloch. Prvý je Control protokol a využíva sa na odosielanie riadiacich správ z MT na server. V tomto protokole sú jednotlivé hodnoty oddelené medzerou. Druhý je Roomba info protokol slúžiaci na posielanie aktuálnych informácií o RPR klientovi. V tomto protokole sú jednotlivé hodnoty oddelené bodkočiarkou.

Poradie dát	Názov	Možné hodnoty	Význam
0	On/Off	0 – 1	Riadiaci povel na začiatok zapínania/vypínania Roomby
1	Y	-180 – 180	Hodnotu náklonu v osi Y
2	Z	-180 – 180	Hodnotu náklonu v osi Z
3	SpeedLimit	0 – 5	Udáva obmedzenie rýchlosti v piatich stupňoch

Tabuľka 3: Control protokol

Poradie dát	Názov	Možné hodnoty	Význam
0	On/Off	0 – 1	Stav Roomby, zapnutá/vypnutá
1	Taking amper	-32 768 – 0	Spotreba Roomby v mA
2	Bumper Left	0 – 1	Stav ľavého nárazníka
3	Bumper Right	0 – 1	Stav pravého nárazníka
4	Clift	0 – 1	Stav skupiny senzorov útesu
5	Batery Capacity	0 – 100	Približný stav batérie Roomby v %

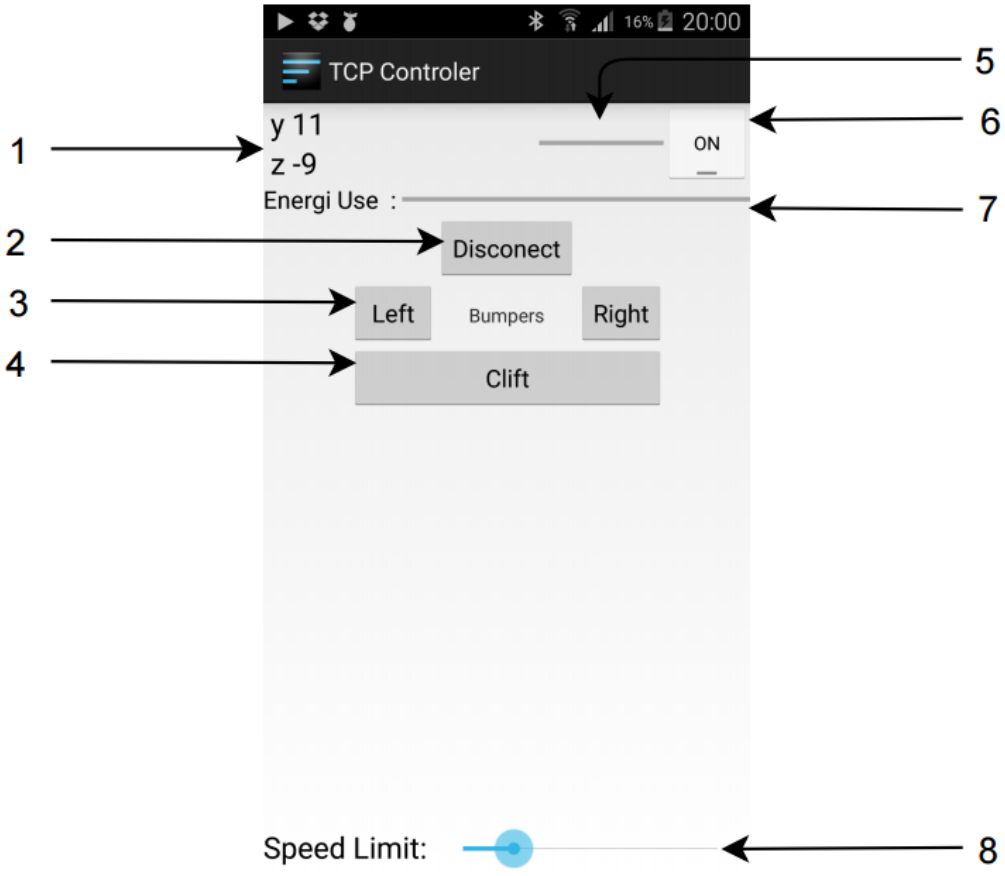
Tabuľka 4: Roomba info protokol

5 Realizácia programu pre Android a Raspberry Pi

V predchádzajúcich kapitolách sa opisujú základné časti, ktoré sú potrebné na riadenie robotického podvozku Roomba pomocou mobilného telefónu s operačným systémom Android. V nasledujúcich kapitolách sa nachádza skompletizovanie jednotlivých častí pre server a pre klienta tak, aby bolo riadenie plne funkčné.

5.1 Implementácia klienta

Robotický podvozok Roomba riadi užívateľ, ktorý po spustení aplikácie na MT ma k dispozícii užívateľské rozhranie na riadenie RPR. Toto rozhranie tvorí trieda, ktorá po spustení nadviaže komunikáciu zo serverom, prípadne indikuje, že server nie je dostupný. Toto užívateľské rozhranie, ktoré je znázornené na obrázku 14, je opísané v nasledujúcej podkapitole.



Obrázok 14: Užívateľské rozhranie Android

5.1.1 Jednotlivé časti užívateľského rozhrania

1. **Zobrazenie aktuálnej orientácie v osiach X a Y**
   Na zobrazenie aktuálnej orientácie MT sa používa komponenta `TextView`, ktorej možno nastaviť zobrazovaný text pomocou metódy `setText(String text);`

2. **Zobrazenie stavu Serveru a Roomby**
   Používa sa komponent Button, ktorá môže nadobudnúť tri hodnoty:
   - **Disconect** – zobrazí sa v prípade, že nie je nadviazaná komunikácia so serverom,
   - **Connected** – zobrazí sa po nadviazaní komunikácie zo serverom a po získaní správy, ktorú opisuje Roomba info protokol,
   - **Roomba On** – zobrazí sa v prípade, že Roomba je v Safe mode.

   Pri zmene hodnôt sa používa metóda `setText(String text).`

3. **Zobrazenie stavu ľavého a pravého nárazníka**
   Tento stav sa indikuje zmenou podfarbenia komponentu Button na červenú farbu, a to metódou `setBackgroundColor().`

4. **Zobrazenie stavu skupiny senzorov indikujúcich útes**
   Podobne ako pri zobrazení stavu ľavého a pravého nárazníka.

5. **Zobrazenie stavu batérie**
   Na toto zobrazenie sa používa komponent `ProgressBar,` ktorý zobrazuje približný stav batérie v percentách; na nastavenie tohto komponentu sa používa metóda `setProgress(int number).`

6. **Tlačidlo na zapnutie a vypnutie Roomby**
   Používa sa komponent TooggleButton, pre ktorý sa vytvorí Listener, a metóda, ktorá po kliknutí užívateľa nastavuje hodnotu tlačidla a indikátora zmien.

```
motorToggleButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if (((ToggleButton) v).isChecked()){
               motorTogle="1";
               senzorChange=true;
               }
        else{
               motorTogle="0";
               senzorChange=true;
        }
    }
});
```

Výpis kódu 6: Nastavenie TooggleButton

7. **Zobrazenie aktuálnej spotreby Roomby**
   Zobrazuje aktuálnu spotreba energie, z ktorej možno odvodiť zmenu povrchu (koberec, hladký povrch) alebo stúpanie povrchu, po ktorom sa Roomba pohybuje za predpokladu, že nebola zmenená rýchlosť. K tomuto zobrazeniu sa používa komponent `ProgressBar` a hodnoty sa nastavujú pomocou metódy `setProgress(int number)`.

8. **Nastavenie rýchlostného limitu**
   Na toto nastavenie sa používa komponent SeekBar, pre ktorý je vytvorený Listener, a metóda `onProgressChanged`, ktorá pri zmene SeekBar nastaví hodnotu `speed` a `senzorChange`.

```
seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {

       @Override
      public void onProgressChanged(SeekBar seekBar, int progress,boolean
fromUser){
        // TODO Auto-generated method stub
        speed=progress;
         senzorChange=true;
      }
});
```

Výpis kódu 7: Nastavenie SeekBar

5.1.2 Spracovanie prijatých dát

Prijímanie dát je opísané v kapitole 4.3.4. *Odosielanie na strane klienta*. Táto kapitola opisuje spracovanie týchto dát a ich zobrazenie na užívateľskom rozhraní, ktoré je opísané v predchádzajúcej kapitole.
Metóda `onProgressUpdate` je volaná po prijatí dát zo serveru, v kóde je ukázaný spôsob spracovania dát, ktoré sú vo formáte, ktorý opisuje Roomba info protokol, a nastavenie komponentov ProgressBar. V tejto metóde sa spracúvajú všetky hodnoty, ktoré sú získané z protokolu.

```
@Override
protected void onProgressUpdate(String... values) {
    super.onProgressUpdate(values);
    try{
            String[] splitValues = values[0].split(";");
            motorPB.setProgress(Integer.parseInt(splitValues[1])*-1);
            bateryPB.setProgress(Integer.parseInt(splitValues[5]));
            . . .

    }
    catch(Exception e){}
}
```

Výpis kódu 8: Spracovanie prijatých dát na strane klienta

5.1.3 Odosielanie na strane klienta

Riadiace signály na strane klienta sú odosielané každú sekundu za predpokladu, že nedôjde k zmene. Ak zmena nastane, nastaví sa parameter `senzorChange` na hodnotu true. Táto hodnota sa kontroluje pomocnou komponentu `Timer`. Tomuto komponentu sa pri konštrukcii určujú dve hodnoty: prvá určuje čas behu a druhá čas do ďalšieho „tiku“. Komponent tiež obsahuje dve metódy: prvú `onTick`, ktorá určuje, čo sa stane po uplynutí doby do ďalšieho „tiku“, a druhú metódu `onFinish`, ktorá určuje, čo sa vykoná po uplynutí času behu. V oboch metódach sa vykonáva odoslanie hodnôt, ktoré sú opísané v Control protokole, a zobrazenie hodnôt X a Y.

```
// Sending in the time
 Timer= new CountDownTimer(1000, 200) {

        public void onTick(long millisUntilFinished) {
                 if(senzorChange==true){
                        String sOrientation = motorTogle +  //power
                                                    " " + oldY +
                                                    " " + oldZ +
                                                    " " + speed ;  //speed limit
                        String syOrientation = " y " + oldY;
                        String szOrientation = " z " + oldZ;

                        yTextView.setText(syOrientation);
                        zTextView.setText(szOrientation);
                        if (mTcpClient != null)
                        {
                                mTcpClient.sendMessage(sOrientation);
                        }
                        senzorChange=false;
                }
        }
        public void onFinish() {

                 //totožní kód ako v metode onTick

                Timer.start();
        }
}.start();
```

Výpis kódu 9: Odosielanie dát serveru

5.2 Implementácia serveru

Server je umiestnený na zariadení Raspberry Pi a zabezpečuje komunikáciu medzi MT a robotickým podvozkom. Roomba je pomenovaná ako ServerRoomba. Bezdrôtová komunikácia medzi MT a Raspberry Pi je opísaná v kapitole 4.3 a komunikácia medzi Raspberry Pi a RPR sa nachádza v kapitole 4.2. Posledným krokom je vytvorenie riadiacej logiky, ktorá zabezpečuje prepojenia medzi týmito dvomi komunikáciami. Táto logika spracováva riadiace signály slúžiace na riadenie pohybu. Zabezpečuje spätnú väzbu, správanie robotického podvozku pri strate spojenia a náraze. Táto logika sa nachádza v nasledujúcich podkapitolách.

5.2.1 Nadviazanie komunikácie

Hneď po spustení aplikácie dochádza k vytvoreniu serveru na porte 8888 a vytvoreniu triedy `RoombaSCI`, ktorá v konštruktore obsahuje vytvorenie sériovej komunikácie s RPR. V prípade, že dôjde k pripojeniu klienta, je vytvorené paralelné vlákno. Toto vlákno sa podrobnejšie opisuje v kapitole 5.2.3 a zabezpečuje spracovanie senzorov RPR. Po prijatí správy od klienta sú hodnoty uložené do premenných `power, y, z` a `SpeedLimit.` Spracovanie týchto hodnôt je opísané v nasledujúcich kapitolách.

```
int main ( int argc )
{
    cout << "Server Start\n";

    int power, y ,z, SpeedLimit;
    int powerOLD = 0;

    pthread_t thr;
    struct Argument a;

  try

    // Create the socket
    ServerSocket server ( 8888 );
    RoombaSCI sci;

    //threead
    a.scirumba = &sci;
    a.delay =100000;

    while ( true )
    {
      server.accept ( new_sock );
      if (pthread_create(&thr, NULL, &vlakno, &a)) { puts("Chyba vlákna");return 1;}

      try
        {
          while ( true )
          {
            //recive data
              new_sock>>recive;
              std::istringstream is(recive);
              is >> power >> y >> z >> SpeedLimit;
     ...
```

Výpis kódu 10: Nadviazanie komunikácie a príjem riadiacich dát

5.2.2 Zapínanie

Nasledujúci kód sa nachádza v metóde `main` a vykonáva sa hneď po načítaní premenných od klienta (pokračovanie Výpis kódu 10: Nadviazanie komunikácie a príjem riadiacich dát). Tento kód sa vykonáva v prípade, že bola zmenená hodnota `power`, ktorá je nastavovaná na strane klienta tlačidlom na vypnutie a zapnutie Roomby zobrazenom na obrázku 15. V prípade, že hodnota `power` je rovná 1, dochádza k volaniu metódy `sci.StartSafe()`. Tá zapne Roombu a uvedie ju do Safe modu. Metódou `sci.Play(1)` indikuje zapnutie Roomby zvukovou sekvenciou.
V opačnom prípade hodnota `power` nadobúda hodnotu 0 a metóda `sci.Play(0)` indikuje vypnutie Roomby. Na dokončenie prehrávania celej zvukovej sekvencie je nutné počkať jednu sekundu. Po sekunde sa Roomba vypína metódou `sci.Power()`. Pred ukončením sa nastavuje aktuálny stav Roomby.

```
...
if(powerOLD != power){
        if(power == 1)
                {
                sci.StartSafe();
                sci.Play(1);
                powerOLD=power;
        }
        else{
                sci.Play(0);
                sleep(1);                       //Zastavenie programu na 1 sekundu
                sci.Power();
                powerOLD=power;
        }
                sci.RoombaON=powerOLD; //Nastavenie aktuálneho stavu Roomba

}
...
```

Výpis kódu 11: Zapínanie Roomby

5.2.3 Ovládanie pohybu

Realizované je pomocou metódy `Drive`, ktorá prepočítava prijaté hodnoty X a Y od klienta. Tieto hodnoty sú v rozmedzí -180° až 180°, ale použitie ovládania v takomto rozsahu je nevhodné a bolo zvolené použitie v rozmedzí 60°. Tento rozsah reprezentujú hodnoty od -30° do 30°. Ďalším parametrom, ktorý metóda spracúva, je `SpeedLimit`. Určuje obmedzenie rýchlosti pohybu RPR v piatich stupňoch. Metóde je odovzdávaná aj vytvorená trieda `RoombaSCI` slúžiaca na komunikáciu s RPR a získanie hodnoty `RoombaON`, ktorá určuje stav vypnutá alebo zapnutá. Celá logika metódy sa vykonáva len v prípade, že je hodnota `RoombaON` rovná `1`.

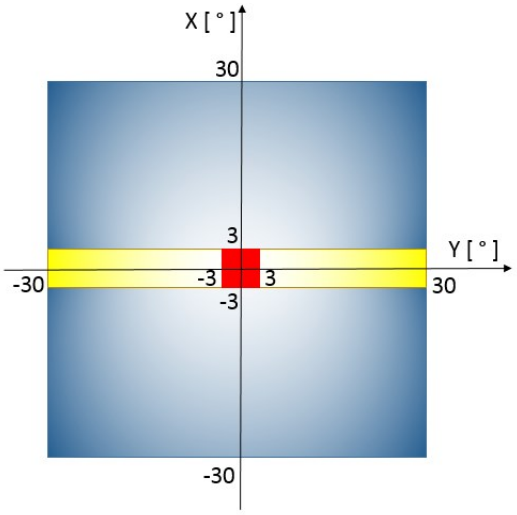
Na vysvetlenie prepočtu náklonu mobilného telefónu na hodnoty odosielané RPR sú použité tri stavy, ktoré sú v grafoch a diagrame aktivít reprezentované farbami:

- **pohyb (modrá)** – reprezentuje pohyb Roomby po priamke alebo po kružnici,
- **otáčanie na mieste (žltá)** – reprezentuje otáčanie Roomby na mieste v smere hodinových ručičiek alebo proti smeru hodinových ručičiek,
- **státie (červená)** – reprezentuje zastavenie Roomby, hodnota `Velocity=0`.

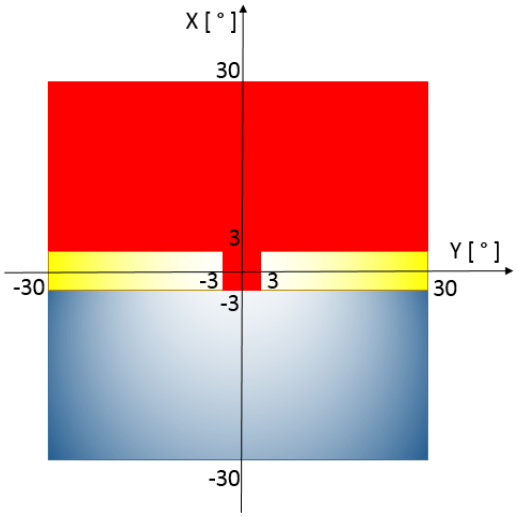
Obrázok 15: Vybrané pohyby Roomby

Na nasledujúcich dvoch stránkach sa nachádza logika prevodu polohy MT na hodnoty, ktoré sú posielané RPR sériovou linkou. Do úvahy sa berú dva spôsoby prevodu v závislosti od hodnôt získaných zo senzorov RPR. Tieto stavy sú zobrazené na obrázku 16 a 17. Prvý zobrazuje správanie, keď senzory neregistrujú žiadnu prekážku. Druhý vychádza z fyzického rozloženia senzorov nachádzajúcich sa v prednej časiť RPR. V tomto prípade dochádza z bezpečnostného hľadiska k zastaveniu pohybu dopredu, a to v prípade detekcie kolízie pravým nárazníkom, ľavým nárazníkom alebo senzorom útesu.

Z grafov pohybu a diagramu aktivít na obrázkoch 16,17 a 18 vyplýva, že spracované hodnoty sa nachádzajú v intervale < -30, 30 >. Hodnoty, ktoré sa nenachádzajú v tomto intervale, sú brané ako maximálna, respektíve minimálna hodnota v danom smere.
Určená je nulová poloha v prípade, že sa X aj Y nachádzajú v intervale < -3, 3 >. Pri tejto polohe dochádza k zastaveniu RPR.
Žltá farba znázorňuje rotáciu na mieste, kde k tejto rotácii dochádza v prípade, že sa hodnota X nachádza v nulovom intervale < -3, 3 > a hodnota Y sa v tomto intervale nenachádza. V tomto prípade je hodnota Radius nastavená na 1 alebo -1 v závislosti od smeru rotácie na mieste. Hodnota Y sa používa na výpočet hodnoty Velocity.
V modrých častiach je hodnota Radius určovaná prepočtom Y a hodnota Velocity určovaná prepočtom X a SpeedLimit.



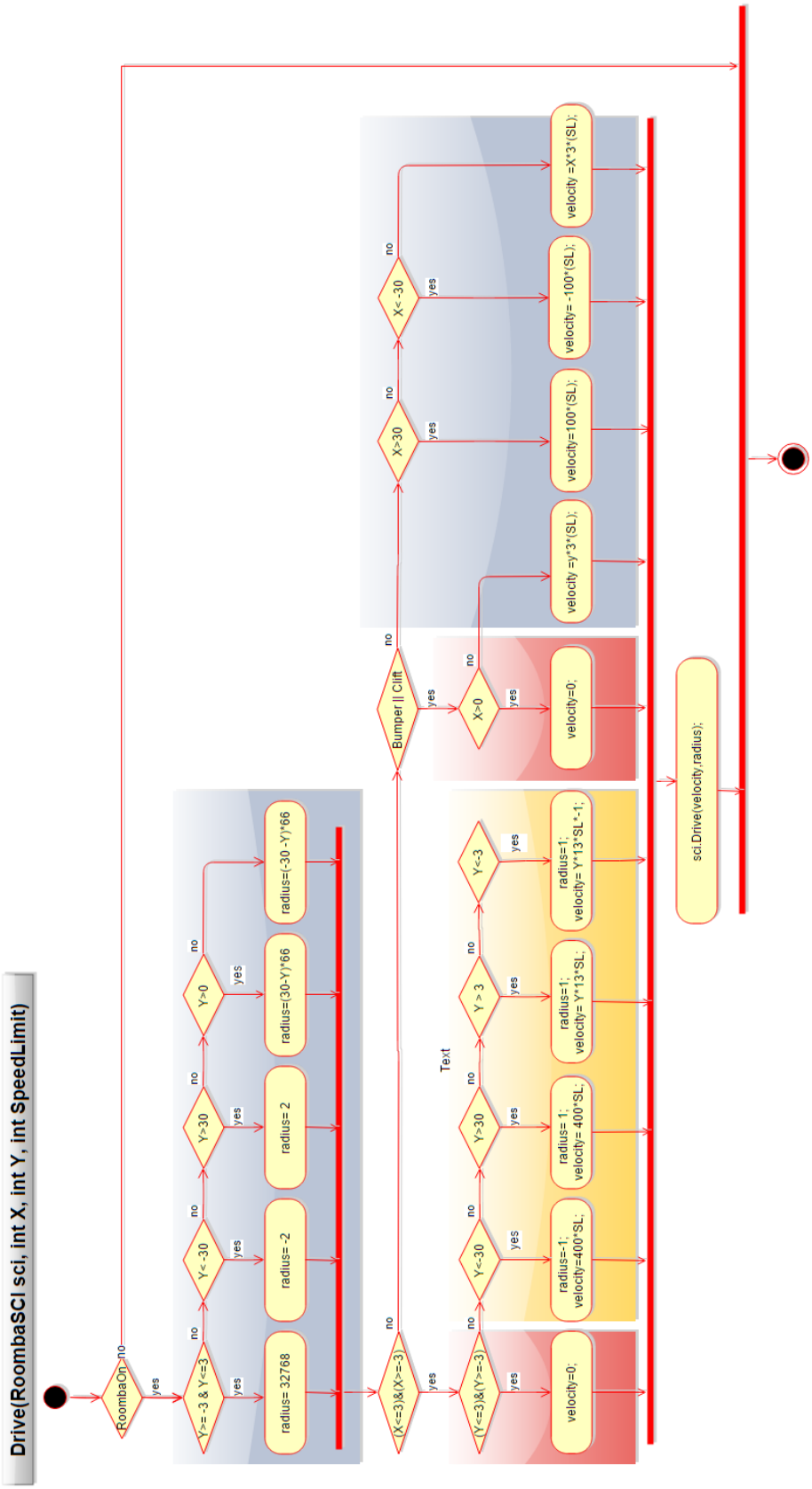
Obrázok 16: Graf pohybu



Obrázok 17: Graf pohybu pri detekcii nárazu alebo útesu

Popis	Poloha z MT		Prepočty		Zasielanie sériovou linkou
	X [°]	Y [°]	Velocity	Radius	Bitové sekvencie
Státie	0	0	0	0	0x89, 0x00, 0x00, 0x00, 0x00
Státie	3	-3	0	0	0x89, 0x00, 0x00, 0x00, 0x00
Rovno dopredu, rýchlo	30	0	500	32 768	0x89, 0x01, 0xf4, 0x80, 0x00
Rovno dozadu, rýchlo	-30	2	-500	32 768	0x89, 0x01, 0xf4, 0x80, 0x00
Rotácia doprava, rýchlo	0	30	500	1	0x89, 0x01, 0xf4, 0x00, 0x01
Rotácia doľava, stredne rýchlo	1	-15	250	-1	0x89, 0x00, 0xfa, 0xff, 0xff
Zatáčanie doprava, rýchlo	30	15	500	1 000	0x89, 0x01, 0xf4, 0x03, 0xe8

Tabuľka 5: Ukážka vybraných prepočtov polohy



Obrázok 18: Diagram aktivít metódy Drive

Funkcie vykonávané v paralelnom vlákne

Vo vlákne sa vykonáva nekonečný cyklus. V prípade, že Roomba nie je zapnutá, odosiela klientovi nulové hodnoty vo formáte protokolu Roomba info. V opačnom prípade za predpokladu, že je sériová linka pre zápis voľná, je volaná metóda `sci->Sensor(),` ktorá získa hodnoty senzorov RPR. Nasleduje volanie metódy `SetLed(sci)` slúžiacej na nastavenie informačného osvetlenia s nasledujúcim významom:

- **modrá dióda** – indikuje Roombu v Safe mode a pripravenú na činnosť,
- **červená varovná dióda** – indikuje náraz alebo výskyt útesu,
- **nastaviteľné pole CLEAN** – slúži na zobrazenie stavu batérie.

Ďalším krokom je zaslanie informácií zo senzorov RPR klientovi a vykonanie sekvencie na jednorazové zastavenie motora v prípade, že sa zistí náraz alebo útes.

Celý nekonečný cyklus sa vykonáva s oneskorením určeným hodnotou `delay`. Táto hodnota je nastavená na 100 000 a určuje tak vykonanie cyklu približne desaťkrát za sekundu.

```
while (1) {
        if(sci->RoombaON== 0 ){
                string ss = "0;0;0;0;0;0";
                new_sock << ss +"\n";
        }
        else {
        if(sci->SendingFree==true){
                sci->Sensor();
                SetLed(sci);
                new_sock << GetRoombaSensor(sci) +"\n";
                //one time stop motor
                if (sci->BumpLeft==true || sci->BumpRight==true || sci-> Clift==true){
                        if(oneTimeStopMotor==false){
                              sci->Drive(0,0);
                              oneTimeStopMotor=true;
                         }
                          else if(sci->BumpLeft==false && sci->BumpRight==false && sci-
>Clift==false){
                                    oneTimeStopMotor=false;
                         }
                }
        }
        }
        usleep(p->delay);
}
```

Výpis kódu 12: Nekonečný cyklus vo vlákne

6 Testovanie odozvy a spoľahlivosti

Na testovanie systému bol použitý robotický vysávač iRobot Roomba, model 581 a mobilné telefóny Samsung Galaxy S4 a S5. Na vytvorenie Wi-Fi komunikácie medzi zariadeniami sa v prevažnej miere používal vytvorený prístupový bod na mobilnom telefóne, ale aj domáca sieť vytvorená Wi-Fi routerom. Na zariadení Raspberry Pi bola nastavená statická IP adresa 192.168.43.82, aby nedochádzalo k prideľovaniu odlišných IP adries DHCP serverom.
Testy prebiehali postupne, ako sa vyvíjali jednotlivé časti aplikácie. V nasledujúcich kapitolách sú uvedené niektoré chyby, ktoré boli zistené počas testovania, a spôsob ich opravy.

6.1 Strata konfigurácie Roomby

Popri testoch bolo zistené, že keď sa vybije batéria Roomby, dôjde k strate všetkých nastavených hodnôt. Hlavná hodnota, ktorá ovplyvní možnosť komunikácie, je rýchlosť sériovej linky. Taktiež sa zistilo, že používaný model nemá prednastavenú hodnotu rýchlosti komunikácie 57 600 B, ale odlišnú. Preto sa použilo manuálne nastavenie rýchlosti seriálovej linky. Po zapnutí Roomby sa drží tlačidlo CLEAN 10 sekúnd. Zmena rýchlosti je nastavená na 19 200 B a je indikovaná tónom. Po tomto nastavení sa použije Aplikácia SETROOMBA, ktorá nastaví najvyššiu možnú rýchlosť sériovej linky a tiež uloží hudobné sekvencie využívané na indikáciu zapnutia a vypnutia.

6.2 Veľké množstvo dát posielaných medzi klientom a serverom

Pri testovaní dochádza k pomalým odozvám RPR alebo pádu serveru. Príčinou je zasielanie informácie o polohe MT pri každej získanej informácii zo senzorov. Preto bolo zavedené oneskorenie, ktoré posiela informácie len päťkrát za sekundu, a to v prípade, že nastala zmena polohy alebo nastavení riadiacich premenných.
Sledovanie senzorov RPR sa vykonáva desaťkrát za sekundu z dôvodu rýchlejšej reakcie na kolíziu alebo útes.

6.3 Prevodník RS-232

Pri testovaní došlo k zničeniu dvoch prevodníkov RS-232, ktoré sa začali prehrievať a prestali komunikovať. Dôvod tejto poruchy nebol zistený a prevodník bol nahradený novým kusom.

6.4 Nepravidelné chyby pri spustení Android klienta

Pri spúšťaní klienta pre Android dochádzalo v niektorých prípadoch k pádu aplikácie. Tento problém nastával pri vytváraní triedy `TCPClient`, ktorej sa predáva odkaz na `MessageListener`. V niektorých prípadoch dochádzalo k tomu, že `MessageListener` ešte nebol vytvorený a nastala chyba, ktorá mala za následok pád aplikácie.
Tento problém sa riešil vytvorením nového konštruktora a metódy `Construct`, a tak sa dával odkaz na `MessageListener` v metóde `connectTask`.

```
// pôvodný konštruktor
public TCPClient(final OnMessageReceived listener)
{
    mMessageListener = listener;
}
//vlastná zmena konštruktora
public TCPClient(){}
public void Construct (final OnMessageReceived listener)
{
    mMessageListener = listener;
}
```

Výpis kódu 13: Konštruktory triedy TCPClient

6.5 Spôsoby testovania

Keďže som nemal prístup k zariadeniam, ktoré by boli schopné zmerať odozvu prejavujúcu sa určitým správaním robotického podvozka Roomba, bolo testovanie vykonávané vizuálnou kontrolou správania RPR a kontrolou rýchlosti zobrazenia informácií zo senzorov (napríklad aktivácia nárazníku) v užívateľskom rozhraní klienta.
Použilo sa aj vypisovanie správania serveru na konzolu a kontrola aktuálneho správania.

7 Záver

Hlavným prínosom tejto diplomovej práce je návrh spôsobu riadenia robotického podvozka Roomba. Na tento cieľ boli vyvinuté dve aplikácie na rôznych platformách, ktoré zabezpečujú komunikáciu medzi sebou prostredníctvom Wi-Fi siete a riadenie robotického podvozku prostredníctvom sériového rozhrania. Tieto aplikácie boli navrhnuté tak, aby ich chod nezaťažoval operačný systém platformy, na ktorom aplikácia beží. Nebola používaná Wi-Fi sieť zahlcovaná zbytočne častou komunikáciou medzi zariadeniami.

S využitím navrhnutých komunikačných protokolov možno vytvoriť klientsku aplikáciu aj na iných platformách a jednoducho tak komunikovať s vytvoreným serverom bežiacim na Raspberry Pi.

Pri testovaní boli zistené určité nedostatky v spracovaní riadiacich signálov, ktoré viedli k úpravám kódu, ale boli tiež zistené vlastnosti robotického podvozku Roomba, ktoré nezodpovedajú špecifikácii Roomba SCI poskytovanej výrobcom.

Raspberry Pi poskytuje veľké možnosti rozšírenia systému, a to pripojenie rôznych snímacích zariadení a vytvorenie takého robotického meracieho zariadenia, prípadne možnosť pripojenia komunikačných zariadení, ako sú monitor, mikrofón a kamera. Počas práce s RPR bola pripojená aj kamera a bolo otestované vytváranie časovo zberných videí a strímovanie obrazu. Na takéto využitie by však bolo vhodné použiť výkonnejšiu verziu Raspberry Pi. Nasledujúci vývoj by mal tiež upraviť užívateľské rozhranie klienta z vizuálneho hľadiska a pridať možnosť zobrazovať video.

Použitá literatúra

[1] UJBÁNYAI, Miroslav. Programujeme pro Android. Vyd. 1. Praha: Grada, 2012, 187 s. Průvodce (Grada). ISBN 978-80-247-3995-3.

[2] KURT, Tod E. Hacking Roomba. Indianapolis. IN: Wiley Pub., c2007, xviii, 436 p. ISBN 90780470072714.

[3] AbcLinuxu: Vyvíjíme pro Android – úvod. [online]. [cit. 2015-04-30]. Dostupné z:http://www.abclinuxu.cz/clanky/vyvijime-pro-android-uvod

[4] Businessworld: Android má konečně své oficiální vývojové prostředí. [online]. [cit. 2015-04-30]. Dostupné z: http://businessworld.cz/aplikace/android-ma-konecne-sve-oficialni-vyvojove-prostredi-12038

[5] Wikipedia: Raspberry Pi. [online]. [cit. 2015-04-30]. Dostupné z: http://cs.wikipedia.org/wiki/Raspberry_Pi

[6] Tutorialspoint: Android Architecture. [online]. [cit. 2015-04-30]. Dostupné z:http://www.tutorialspoint.com/android/android_architecture.htm

[7] Microsoft TechNet: Model protokolu TCP/IP. [online]. [cit. 2015-04-30]. Dostupné z: https://technet.microsoft.com/cs-cz/library/cc786900(v=ws.10).aspx

[8] TOUGHER, Rob. The Linux Documentation Project: Linux Socket Programming In C++. [online]. 2002 [cit. 2015-04-30]. Dostupné z: http://tldp.org/LDP/LG/issue74/tougher.html

[9] MARAVITSAS, Nikos. Java Code Geeks: Android Socket Example. [online]. 2013 [cit. 2015-04-30]. Dostupné z: http://businessworld.cz/aplikace/android-ma-konecne-sve-oficialni-vyvojove-prostredi-12038

[10] R. SWEET, Michael. University of Minnesota: Serial Programming Guide for POSIX Operating Systems. [online]. 2013 [cit. 2015-04-30]. Dostupné z: https://www.cmrr.umn.edu/~strupp/serial.html#1

[11] ŠKOPEK, Pavel. Mobilenet: Techbox: váš telefon je prošpikovaný senzory [online]. [cit. 2015-04-30]. Dostupné z: http://mobilenet.cz/clanky/techbox-vas-telefon-je-prospikovany-senzory-12496

[12] Mobilmania: Tříosý gyroskop v mobilu blízké budoucnosti [online]. [cit. 2015-04-30]. Dostupné z: http://www.mobilmania.cz/triosy-gyroskop-v-mobilu-blizke-budoucnosti-co-umi/a-1123872/default.aspx

[13] IRobot: Open Interface [online]. [cit. 2015-05-01]. Dostupné z: http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf

Adresárová štruktúra priloženého CD

/android_apk	Adresár obsahuje inštalační balík pre OS Android
/android_crc	Adresár obsahuje zdrojové kódy aplikácie pre Android
/doc	Adresár obsahuje dokumentáciu Diplomovej práce
/server_src	Adresár obsahuje zdrojové kódy Serveru riadiaceho RPR

Prílohy

Príloha 1: Kompletné zapojenie RPR